Agent Technology from a NASA Perspective

Walt Truszkowski, Harold Hallock
{Walt.Truszkowski, Harold.L.Halloch.1}@gsfc.nasa.gov
NASA Goddard Space Flight Center, Maryland, USA

Remote Agent Team
James Kurien Point-of-Contact
kurien@ptolemy.arc.nasa.gov

Abstract

NASA's drive toward realizing higher levels of autonomy, in both its ground and space systems, is supporting an active and growing interest in agent technology. This paper will address the expanding research in this exciting technology area.

As examples of current work, the Lights-Out Ground Operations System (LOGOS), under prototyping at the Goddard Space Flight Center (GSFC), and the spacecraft-oriented Remote Agent project under development at the Ames Research Center (ARC) and the Jet Propulsion Laboratory (JPL) will be presented.

1. Introduction

This paper presents a discussion of two NASA applications of agent technologies that are being used to support the realization of both ground system and spacecraft autonomy. The LOGOS, which is under prototyping at GSFC, will enable researchers to evaluate the performance and effectiveness of a community of agents operating in a ground system environment. The maturer ARC/JPL Remote Agent project will demonstrate the effectiveness of an agent in partial control of a spacecraft. Each of these domains brings differing resources and constraints that need to be taken into consideration when designing and implementing a participating agent. Working in both the ground system and spacecraft domains is providing NASA with an opportunity for developing a comprehensive agent technology.

2. Background

To establish a context for later discussions, we provide in this section a brief summary and analysis of the responsibilities and functionality of traditional ground and flight systems as viewed from the framework of a total system process, followed by a highlighting of the key drivers when making flight-ground trades. This analysis

was done from a near-Earth unmanned scientific satellite system perspective. However, the analysis would mostly carry over to deep space missions.

2.1 Introduction to Ground System Software

Traditionally, the ground system has been almost exclusively responsible for spacecraft planning & scheduling, establishment of communications and (in some cases) processing. The ground has also shouldered the bulk of the calibration burden (both science and engineering) and much of the job of spacecraft Health & Safety (H&S) verification. And when major onboard anomalies/failures arise, ground-based Flight Operations Team (FOT) personnel were charged with determining the underlying cause and developing a long-term solution.

So the traditional ground system has occupied the ironic position of having most of the responsibility for managing the spacecraft and its activities, and yet (with the exception of the planning & scheduling function) relying on the spacecraft to provide nearly all information required for carrying out those responsibilities. Today, in a more conventional workplace setting, this kind of work organization might be analyzed (from a reengineering perspective) to be an artificially fragmented process with unnecessary management layering leading to degraded efficiency and wasteful costs.

From the perspective of reengineering, the standard solution to this type of problem is to re-distribute the processes to empower the local sites, e.g. the spacecraft where most of the information originates, to have more responsibility in the handling of that information. To achieve this type of reengineering there needs to be trades made between ground system functionality and flight system functionality.

A prerequisite to performing these flight-ground trades is to identify all the components of the spacecraft operations process, initially without regards to whether that component is performed onboard or on the ground. The following is a break down of operations into a set of activities. The order of the activities is roughly increasing in time relative to the end-to-end operations process, from defining spacecraft inputs to utilizing spacecraft outputs, although some activities (such as fault detection and correction) are continuous and in parallel with the main line.

1. Planning & scheduling
2. Command loading (including routine table uplink)
3. Science Schedule Execution
4. Science Support Activity Execution
5. Onboard Engineering Support Activities (housekeeping, infrastructure, ground interface, utility support functions, onboard calibration, etc.)
6. Downlinked Data Capture
7. Data and Performance Monitoring

8. Fault Diagnosis
9. Fault Correction
10. Downlinked Data Archiving
11. Engineering Data Analysis/Calibration
12. Science Data Processing/Calibration

Currently, many of the activities listed above are partially automated (for example, planning & scheduling and data and performance monitoring), and may well be fully autonomous (either within the ground or flight systems) in the next 10 years. Some of these functions are already largely performed autonomously onboard. We now give a brief description of each of the operations.

2.1.1 Planning and scheduling

Especially for Low Earth Orbit (LEO) GSFC missions, the ground system planning and scheduling function traditionally has been responsible for generating a detailed and optimized timeline of spacecraft activities. Sometimes (as in the case of Hubble Space Telescope (HST)) this is based on rather complex predictive modeling of the spacecraft's environment and anticipated behavior. The ground system recasts (and augments) the timeline information in an appropriate manner for processing on the spacecraft. The timeline is then executed onboard via a (largely) time driven processor. Often along with the nominal and expected timeline, the ground interleaves in it a large array of alternate branches, to be executed in place of the nominal timeline in the event that certain special conditions or anomalies were encountered. So the resulting product could be a highly coupled, time dependent mass of data which, in the past, occupied a substantial fraction of available onboard storage.

Ironically, the ground system's creation of the timeline data block itself could be almost as highly a time-dependent process as the execution of actual timeline onboard. HST provides a particularly complex example. Long-term scheduling (look-ahead intervals of several months to a year) are used to block-out accepted proposal targets within allowed geometric boundary conditions. The geometry factors are typically dominated by Sun angle considerations, with additional contributions from issues such as moon avoidance, maximizing target orbital visibility, obtaining significant orbital dark time or low sky brightness, and meeting linkages between observations specified by the astronomer.

On the intermediate term (a few weeks to a couple of months), LEO spacecraft targets are ordered and scheduled relative to orbital events such as South Atlantic Anomaly (SAA) entrance/exit and Earth occultation's, and the duration of their associated observations (based on required exposure time computations) were estimated. Concurrently, support functions requiring scheduling, like communications, are interleaved with the science target scheduling.

Lastly, on the short term (a day to a week), final detailed scheduling (both of science targets and support functions) to a precision of seconds is performed using the most accurate available model data (for example, the most recent predicted spacecraft ephemeris), and with the possibility for inclusion of new targets (often referred to as Targets of Opportunity (TOO) not previously considered.

At times the software needed to support the intermediate and short term scheduling processing performed by the ground system has been massive, complex, and potentially very brittle. Further, multiple iterations of the process, frequently involving a lot of manual intervention (at considerable expense), can be required to produce an error-free schedule. Although considerable progress has been made in streamlining this process and reducing its associated costs, the mathematical modeling remains fairly sophisticated and some amount of operational inefficiency is inevitable due to the necessity of relying on approximations during look-ahead modeling.

2.1.2 Command Loading

By contrast to the planning and scheduling function, command loading is quite straightforward. It consists of translating the directives output from planning and scheduling (plus any real-time directives, table loads, etc. generated at and output from the control center) into the language/formats understandable by the flight computer and compatible with the communications medium. As communications protocols and the input interfaces to flight computers become more standardized, this ground system function will become steadily more automated via Commercial Off-the-Shelf (COTS) tools.

2.1.3 Science Schedule Execution

Science schedule execution refers to all onboard activities that directly relate to performing the science mission. They include target acquisition, Science Instrument (SI) configuration, and SI operations on targets (for example, exposure time management)

2.1.4 Science Support Activity Execution

Science support activities are those specifically performed to ensure the success of the science observation, but are not science observations themselves, nor are they routine housekeeping activities pertaining to maintenance of a viable observing platform.. They are highly mission/SI specific activities and may include functions such as Optical Telescope Assembly (OTA) calibration and management and SI direction of spacecraft operation (such as pointing adjustment directives). These activities may be performed in immediate association with ongoing science, or may

be performed as background tasks disjoint from a current observation. Although executed onboard, much (if not all) of the supporting calculations may be done on the ground and the results uplinked to the flight computer in the form of tables or commands.

2.1.5 Onboard Engineering Support Activities

Onboard support activities are routine housekeeping functions pertaining to maintenance of a viable observing platform. The exact form of their execution will vary from spacecraft to spacecraft, but general categories are common within a mission type (e.g..., GEO Earth-pointer, LEO celestial-pointer, etc.). Housekeeping functions include angular momentum dumping, data storage & management, attitude & orbit determination and/or prediction, attitude control, and orbit station keeping. These activities may be performed in immediate association with ongoing science, or may be performed as background tasks disjoint from a current observation. .Just as with science support activities, some of the supporting calculations may be done on the ground and the results uplinked to the flight computer in the form of tables or commands.

2.1.6 Downlinked Data Capture

Capture of downlinked telemetry data is rather straightforward and highly standardized. This ground system function will become steadily more automated via COTS (Commercial Off The Shelf) tools.

2.1.7 Data and Performance Monitoring

Monitoring of spacecraft performance and H&S by checking the values of telemetry points and derived parameters is a function that is currently shared between flight and ground systems. While critical H&S monitoring is an onboard responsibility (especially where triggers to safemode entrance are concerned), the ground, in the past, has performed more long term, non-real-time quality checking, such as hardware component trending, accuracy analysis, as well as treatment of more general performance issues (e.g., overall observing efficiency).

2.1.8 Fault Diagnosis

Traditionally prior to launch, the system engineers would identify a whole host of key parameters that need to be monitored onboard, specify tolerances defining in-range vs. out-of-range performance, and identify FSW responses to be taken in real-time and/or FOT responses to be taken in near-real-time. To do this the system engineers have started with a set of failure scenarios in mind, identified the key parameters (and their tolerances/thresholds) that would measure likely symptoms of those failures, figured out how to exclude possible red-herrings (i.e., different

physical situations that might masquerade as the failure scenario under consideration), and (in parallel) developed corrective responses to deal with those failures. So the process of transitioning from the symptoms specified by the parameters to the correction action (often a static table entry) that constitutes the diagnosis phase conceptually actually occurs (pre-launch) in the reverse order and the intellectual output of the process is stored onboard.

In the post-launch phase, the system engineers/FOT may encounter an unanticipated problem and must perform a diagnosis function using the telemetry downlinked to the ground. In such cases, operations personnel must rely on their experience (possibly with other spacecraft) and subject matter expertise to solve the problem. When quick solutions are achieved, the process often used is that of pattern recognition (or, more formally, case-based reasoning as will be discussed later), i.e., the recognition of a repetition of clues observed previously when solving a problem in the past. Failing at the pattern recognition level, a more lengthy general analytical phase (the human equivalent of state modeling) typically ensues that is manually intensive and at times very expensive.

2.1.9 Fault Correction

Currently, generating a plan to correct an onboard anomaly, fault, or failure is exclusively a ground responsibility. These plans may be as simple as specification of a mode change or as complex as a major hardware reconfiguration or a FSW code modification. In many cases, canned solutions are stored onboard for execution in response to an onboard trigger or ground command, but creation of the solution itself was done by ground system personnel, either in immediate response to the fault or (at times) many years prior to launch in anticipation of the fault. And even where the solution has been worked out and validated years in advance, conservative operations philosophy has often retained within the ground system power to initiate the solution.

2.1.10 Downlinked Data Archiving

Archiving of downlinked telemetry data (including in some cases distribution of data to users) is rather straightforward and highly standardized. This ground system function will become steadily more automated via COTS tools.

2.1.11 Engineering Data Analysis/Calibration

Traditionally, nearly all spacecraft engineering analysis and calibration functions (with the exception of gyro drift bias calibration and, for the Small Explorer (SMEX) missions, magnetometer calibration) have been performed on the ground. These include attitude sensor alignment and polynomial calibrations, battery depth-of-

discharge & state of charge analyses, communications margins evaluations, etc. Often the work in question has been iterative and highly manually intensive.

2.1.12 Science Data Processing/Calibration

Science data processing and calibration have been nearly exclusively a ground system responsibility for two reasons. First, the low computing power of radiation hardened onboard computers, relative to that available in ground systems, has limited the degree to which science data processing can be performed onboard.. Second, the science community generally has insisted that all the science data be brought to the ground. Their position arises from a concern that the data might not be processed as thoroughly onboard as it might on the ground, and that the science data users often process the same data multiple times using different algorithms, calibrations, etc., sometimes years after the data was originally collected.

2.2 Introduction to FSW (Flight Software)

Although highly specialized to serve very precise (and often mission-unique) functions, FSW must simultaneously satisfy a surprisingly broad spectrum of competing needs.

First, it is the FSW that provides the ground system an interface with the flight hardware, both engineering and science. Since these hardware components are constantly being upgraded as their associated technologies continue to advance, the FSW elements that "talk" to the hardware components must regularly be updated as well. Fortunately, as the interface with the ground system (at least here at GSFC) has largely been standardized, the FSW elements that talk to the ground remain largely intact from mission to mission. It is in fact the ability of the FSW to mask changes in flight hardware input/output channels that has provided the ground system a relatively stable environment for the development of standardized COTS products, which in turn has enabled dramatic reductions in ground system development costs.

Second, it is the responsibility of the FSW to respond to events that the ground system cannot deal with, because

1. The spacecraft is out of contact with the ground.
2. The response must be immediate.
3. Critical spacecraft or payload H&S issues are involved.
4. The ground lacks key onboard information for formulating the best response.

Historically, the kinds of functions allocated to FSW for these reasons were ones such as the Attitude Control Subsystem (ACS), safemode processing and transition logic, fault detection and correction, target acquisition logic, etc.

Third, the FSW can be used to streamline (at least in part) those (previously considered ground system) processes where an onboard, autonomous response is cheaper or more efficient. In many of these cases, routine processes may first be performed manually by operations personnel, following which automated ground software is developed to perform the process. After the automated ground process has been fully tested operationally, the software (or algorithms) may then be migrated to the flight system.

Fourth, the process may be performed onboard in order to reduce demand on a limited resource. For example, downlink bandwidth is a valuable, limited quantity on most missions, either because of size/power constraints on spacecraft antennas/transmitters, or because of budget limitations on the size of the ground antenna. In such cases, FSW may be used to compress the output from payload instruments or prune excessive detail from the engineering telemetry stream to accommodate a smaller downlink volume.

As can be seen from even casual consideration of these few examples, the demands placed on FSW are of a widely varying nature. Some require high precision calculation of complex mathematical algorithms. These calculations often must be performed extremely quickly and the absolute time of the calculation must be accurately placed relative to the availability of key input data (i.e., the data latency issue). On the other hand, some FSW functions must process large quantities of data (for example, compression of science instrument output), or must store and manage the data. Other functions must deal with intricate logic trees and orchestrate real-time responses to anomalies detected by self-monitoring functions. And because the FSW is the key line of defense protecting spacecraft H&S, all these functions must be performed flawlessly and continuously, and often (due to onboard processor limitations) must be tightly coupled in several processing loops.

The following is a list of the traditional FSW functions:

1. Attitude Determination and Control
2. Sensor Calibration
3. Orbit Determination/Navigation (traditionally orbit maneuver planning a ground function)
4. Propulsion
5. Executive
6. Command Processing (target scheduling traditionally ground function)
7. Engineering and Science Data Storage

8. Communications
9. Electrical Power Management
10. Thermal Management
11. Science Instrument Commanding
12. Science Instrument Data Processing
13. Data monitoring (traditionally no trending)
14. Fault Detection, Isolation and Recovery (FDIR)
15. Safemode (separate ones for spacecraft and payload instruments)

2.2.1 Attitude Determination and Control, Sensor Calibration, Orbit Determination, Propulsion

Often the first 4 functions reside within a separate Attitude Control System (ACS) processor because of the high Central Processing Unit (CPU) demands of its elaborate mathematical computations. Item 1 includes the control laws responsible for keeping the spacecraft pointing in the desired direction and reorienting the spacecraft to a new direction. Currently, onboard attitude sensor calibration is limited to gyro drift bias calibration (and for some spacecraft a coarse magnetometer calibration).

Orbit determination may be accomplished by measurement (Global Positioning System (GPS) for example), solving the equations of motion, or by use of an orbit propagator. Traditionally, orbit maneuver planning has been the responsibility of the ground, but some experiments will be performed migrating routine stationkeeping maneuver planning onboard Earth Orbiter-1 (EO-1). Regardless whether the orbit maneuver planning is done onboard or on the ground, the onboard propulsion subsystem has responsibility for executing the maneuvers via commands to the spacecraft's thrusters, which also at times may be used for attitude control and momentum management.

2.2.2 Executive, Command Processing, Engineering and Science Data Storage, Communications

The Command & Data Handling (C&DH) processor includes the next 4 functions, and may (depending on the flight system design) include many of the remaining ones. The executive is responsible for coordinating and sequencing all the onboard processing, and separate local executives may be required to control lower level processing within a subsystem. The command processor manages externally supplied stored or real-time commands, as well as internally originated commands to spacecraft sensors, actuators, etc. Again, depending on the design, some command management may be under local control.

The C&DH also have management responsibility for engineering and science data storage (in the past via tape recorders but nowadays usually via solid state storage).

Depending on the level of onboard sophistication, much of the bookkeeping job may be shared with the ground, although the trends are towards progressively higher levels of onboard autonomy. Telemetry uplink and downlink are C&DH responsibilities as well, although articulation of moveable actuators (such as high gain antenna gimbals) as well as any supporting mathematical modeling associated with communications (e.g., orbit prediction) are typically the province of the ACS.

2.2.3 Electrical Power Management, Thermal Management SI Commanding, SI Data Processing

Critical H&S functions like spacecraft electrical power and thermal management is usually treated as separate subsystems, although the associated processing may be distributed among several physical processor locations depending on the design of the flight system. This distribution of sub-functionality is particularly varied with regards to science instrument (SI) commanding and data processing given the steadily increasing power of the SIs associated microprocessors. Currently, we are including any onboard processing associated with a spacecraft's Optical telescope Assembly (OTA) within the context of the SI functions.

2.2.4 Data Monitoring, FDIR

The processing associated with the next two functions (data monitoring and FDIR) are even more highly distributed. Typically, the checking of individual data points and the identification of individual errors (with associated flag generation) is done locally, often immediately after the measurement is read out from its sensor. On the other hand, fault recovery is typically centralized so responses to multiple faults can be dealt with in a systematic manner.

2.2.5 Safemode

The last item, safemode, may include several independent sub-functions, depending on the cost and complexity of the spacecraft in question. Typical kinds of safemode algorithms include Sun acquisition modes (to maintain power positive, healthy thermal geometry, and protect sensitive optics), spin-stabilized modes (to maintain attitude stability), and inertial hold mode (to provide minimal perturbation to current spacecraft state). Usually, the processing for one or more of these modes is located in the ACS processor, but there also usually is a fallback mode in a special safemode processor in case the ACS processor itself has gone down. The individual SIs themselves also have separate safemode capabilities, and anomalies causing the C&DH processor to become unavailable are dealt with via a special uplink-downlink card that in the absence of the C&DH processor enables continued (though limited) ground communication with the spacecraft.

2.3 Flight vs. Ground Implementation

Recently expanding levels of onboard autonomy have been enabled by growth in flight data system capacities (CPU, Input/Output (I/O), storage, etc.), as well as the new approaches to and structures for FSW design and development (object-oriented design, expert systems, remote agents, etc.). In particular, operational activities that previously were the private domain of the ground systems (such as planning & scheduling, engineering data analysis and calibration, and science data processing and calibration) now provide exciting opportunities for shifting responsibility from the ground to the flight component.. This would allow taking advantage of the unique strengths inherent in a real-time software system in direct contact with the flight hardware.

The key advantages possessed by the flight component over the ground component are immediacy, currency, and completeness. Only the flight component can instantly access flight hardware measurements, process the information, and respond in real-time. For example, for performance of basic spacecraft functions such as attitude control and thermal/power management, only the FSW has direct access in real-time to critical information needed to define the spacecraft's operational state, as well as the direct access to the spacecraft actuator hardware required to create and maintain the desired state. The FSW is also the only component of the integrated flight/ground operational system with full-time access to all relevant information for applications such as fault detection and Science Instrument (SI) target acquisition.

By contrast, in the past, the advantage of the ground over the flight segment has been the larger, more powerful ground computers that (for example) have enabled the ground system to execute extremely intricate schedule optimization algorithms using highly complex predictive models. However, as the power of the flight computers continues to grow with time, a partial shift of even of these traditional ground monopolies may be justified to take advantage of the real-time information exclusively available onboard. In fact, as this hardware differences between the two platforms environments narrow, the distinction between flight-based vs. ground-based may begin to blur.

In conjunction with ground/space trades, it is an interesting exercise to contemplate what happens to information as it migrates from the spacecraft to the ground system. As Figure 2.1 suggests, in a simplified way, there occur several abstractions that can affect decision-making. The origin of the information is the spacecraft that is the real entity that is the subject of the decision-making process. This information is downloaded to the ground in a raw data format (sequences of zeros and ones) and is stored in a computer memory and converted to what are termed “engineering units”... These engineering units form the basis of the information that will be presented to an operator or analyst . The conversion from raw data to engineering units is a mapping, which introduces a level of abstraction. After this conversion the information is still in the computer. The act of presenting it to an operator or analyst

, using perhaps some graphical representation, introduces yet another abstraction. The final abstraction process comes when the operator or analyst interprets what he sees with reference to his internal mental model of the spacecraft and its operations.

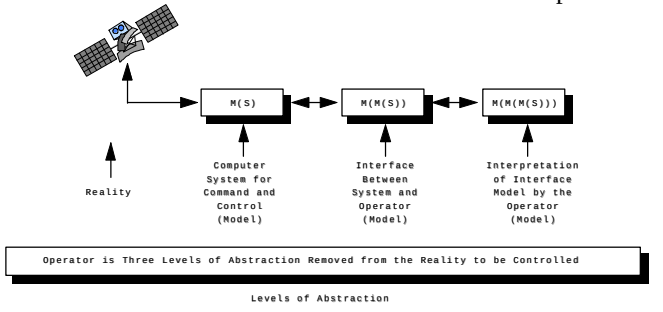

Figure 2.1. Levels of Abstraction

At each level of abstraction there is the possibility of loosing information, adding information or misinterpreting information. Moving the handling and interpretation of the information closer to the source (the spacecraft) will help alleviate the possibility of errors introduced by the abstraction processes. The introduction of agent technology both on the ground and in space is a step in this direction.

3. LOGOS (Lights-Out Ground Operations System)

This section introduces the concept of agents and related concepts as they are currently understood in the Goddard Space Flight Center (GSFC) context, and discusses the LOGOS (Lights-Out Ground Operations System) which is an environment for demonstrating and evaluating the full spectrum of agent-based capabilities required for comprehensive ground/space systems automation.

3.1 Agents in the Goddard Context

In the literature there are many well-documented attempts to define what an agent is. The intent here is not to add confusion but to construct a definition of agent that is meaningful in the context of currently envisioned NASA/Goddard applications.

Analysis of the role of human operators in current ground system centers has contributed to our understanding of the mandatory and desirable characteristics of agents (or the emergent behaviors of an organization of agents) needed to realize ground system autonomy. A near-term objective of ground system autonomy is to achieve a complete and comprehensive realization of "lights-out" operations. Lights-out operations are just that - operation of a ground control center without the presence or direct intervention of people. Essentially, the agents "replace" in some sense the operators and assume the responsibility for their control center activities.

In the current understanding of lights-out operations, human operator intervention is not totally precluded but is supported through operator dialog with an agent or group of agents on an as-needed basis.

The ground system autonomy focus imposes unique requirements on the agents that will be involved in achieving its autonomy. The position we take is that there is a spectrum of agents -- from a minimal agent (reactive) to an intelligent agent (deliberative and social) -- that is needed to support the activities in this domain. In our analysis of ground system autonomy , we have identified the following as typical capabilities that need to be supportable by an agent or a community of agents:

- Ability to initialize a ground system in preparation for a spacecraft contacts.
- Ability to utilize ground system resources to accomplish a goal.
- Ability to maintain operations during a “normal” contact.
- Ability to effectively respond to a fault situation.
- Ability to provide closed-loop support for spacecraft commanding.
- Ability to interact with the outside world in emergency situations.
- Ability to document all ground system operational events.
- Ability to shut down the ground system when not needed in full operational mode.

Analyses of these capabilities have given us a good start at establishing a conceptual understanding of what an agent is and what levels of intelligence it needs to posses. This is addressed in the next subsection.

3.2 Introduction to Agent Concepts

With an understanding of the types of activities that ground control personnel engage in and the prospect of integrating their functionality into the ground software system (via agents), an effort to define the relevant concepts associated with agents was undertaken. Table 3.1 provides a capsule view of the results of that effort. [1]

Table 3.1. Agent Concepts and Definitions/Discussions

Agent Concept	Definition/Discussion
Perception	A software process is said to perceive whenever it obtains data/information from a source in its environment, including itself and communication with other agents. The results of perception are percepts.
Autonomy	A software process is autonomous whenever it acts to achieve a goal according to its own percepts, internal states and knowledge. Inherent is the concept of reasoning.
Competency	A particular capability that an agent possesses

Basic Software Agent	A software process that perceives, is autonomous and has one or more goal-reaching competencies
Attribute	An inherent characteristic or quality that reflects or contributes to, in some way, a capability.
Attribute Understanding	Ability or an agent to modify its mode of communication with others based on an understanding of their communication capabilities.
Adaptable Communication	Ability of an agent to modify its mode of communication with others based on an understanding of their communication capabilities.
Planning	Ability to formulate steps needed to achieve a goal.
Replication	Ability of an agent to clone itself when required to achieve a goal within certain constraints
Temporal Understanding	Ability to understand and reason about time in relation to goal achievement.
Locational Understanding	Ability to understand and reason about self location and the location of others that may be required to support the execution of a goal-achieving plan.
Learning	Ability to increase and/or improve competencies.
Intelligent Agent	An agent is intelligent to the degree that it possesses attribute understanding, adaptable communication planning, replication, temporal understanding, locational understanding a learning capabilities.

Real-world multi-agent systems (like LOGOS to be discussed in 3.3) are designed to effectively employ agents with widely differing levels of intelligence. There is no mandate that all agents in a given agent-based system should have the same level of intelligence. Based on our analysis we feel that in multi-agent systems it is to be expected that the agents would either possess differentiated skills for handling different parts of the application problem or possess the capacity to learn in a nontrivial way (by which it is conceivable that the agent's level of intelligence would be modified).

One important agent-related concept that was adopted for use in the LOGOS relates to the manner in which an agent accomplishes a job. The approach taken essentially amounts to considering an agent as a tool user. The following Figure 3.1 graphically illustrates the idea.

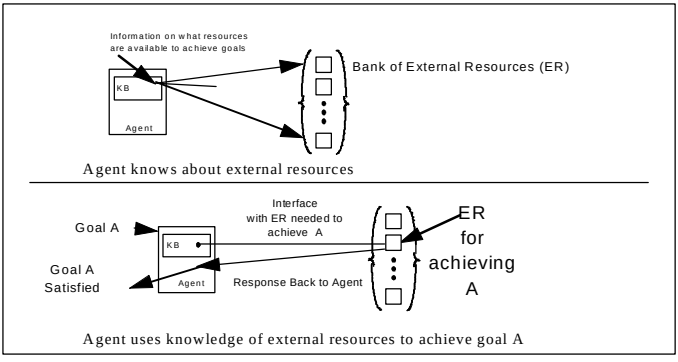

Figure 3.1 An Agent Using an External Resource to Accomplish a Goal

As an example of this concept consider the following. In a control center software system there typically exists an expert system that monitors telemetry from a spacecraft subsystem and reports anomalies to an operator along with suggested corrections if possible. In our approach to agent-based automation of the control center the expert system resource could still be used if the agent knew about it and knew how to interact with it and interpret its results. The monitoring and advise-giving mechanisms would not have to be an integral part of the agent's knowledge base. The agent would have to know how to interpret this information and act on it as would a human operator. In LOGOS there are many examples of this concept in action.

Before considering the LOGOS concept let us consider, for one more time, the idea of an agent as a surrogate controller. Figure 3.2 illustrates that in a lights-out system environment there are several layers of automation. Agent technology will be required to provide the higher-levels of automation required to compensate for the lack of human presence in the lights-out environment

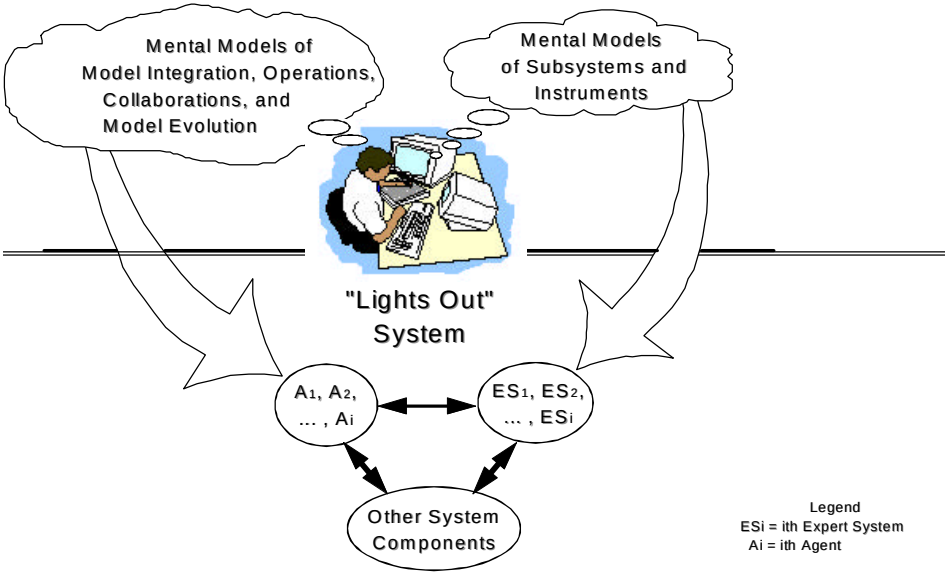

Figure 3.2. View of Operator Integration into a Lights-Out System

As the figure illustrates, the monitoring of spacecraft subsystems is an activity that can and is readily handled by an expert system. However activities such as coordination among a group of operators are at a higher intellectual level which, we feel, require the introduction of agent technology to realize.

3.3 LOGOS (Lights-Out Ground Operations System)

In the development of the initial LOGOS prototype [2] it was decided to distribute the needed operational functionality among a group of agents in a community rather than develop a single monolithic agent capable of all of the needed functionality. Among other things, this approach allows us to research such concepts as the emergence of higher level capabilities from the cooperation of agents with less comprehensive capabilities.

As depicted in Figure 3.3 illustrates the overall operating philosophy for LOGOS is quite straightforward. The agent community of LOGOS attempts to handle any spacecraft anomaly that is not handled by the pass automation process. In the event that it cannot, it establishes an interaction with a remote analyst for help. In doing so it needs to provide to the analyst sufficient contextual information and appropriate representations of what it considers to be the anomaly in order to enable the analyst to successfully address the problem. As the analyst interacts with the community in the solution of the problem the community as whole, as well as individual agents, can learn from the experience. (This particular goal has yet to be achieved.)

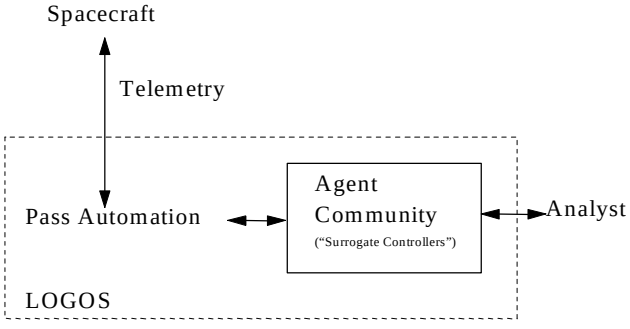

Figure 3.3. The LOGOS Concept

Figure 3.4 illustrates the first version of the LOGOS architecture (currently LOGOS is under redesign to bring about needed enhancements). As the spacecraft sends down telemetry it is formatted and stored in the TPOCC (Transportable Payload Operations Control Center) data server. A GenSAA (Generic Spacecraft Analyst Assistant)/Genie ((Generic Inferential Executor) application, which basically is an expert system, accesses data from this server in its function of executing pass plans and monitoring the health and safety of the spacecraft. Mission Operations Planning and Scheduling System (MOPSS) is the source of pass plan details and is used by Genie to fabricate a pass script for automating pass activities. Each of these resources is associated with an Interface Agent in the LOGOS agent community. Agents in the community will be addresses shortly

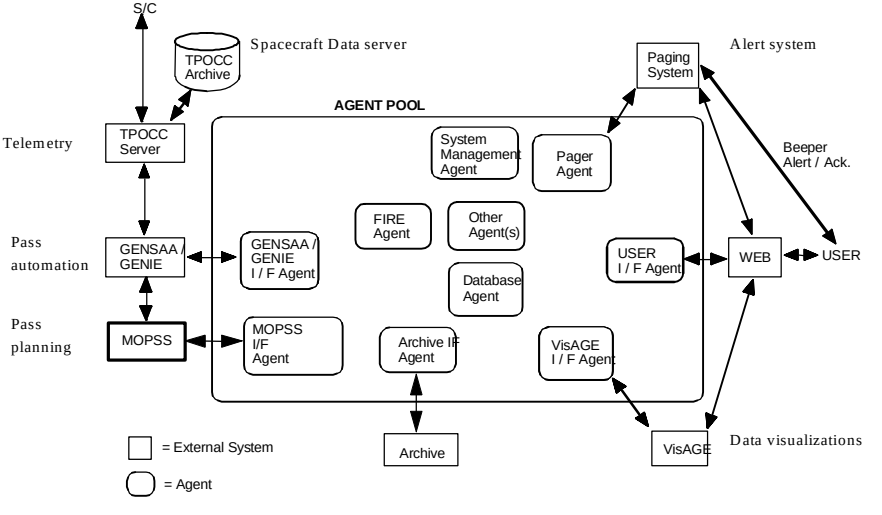

Figure 3.4. Lights-out Ground Operations System (LOGOS) Architecture

The development of LOGOS has been done in the context of object-oriented technology. The implementation of the various agents has been realized in Java. Figure 3.5 give an overview of the agent class hierarchy in LOGOS. Table 3.2 describes the agent class definitions for the LOGOS.

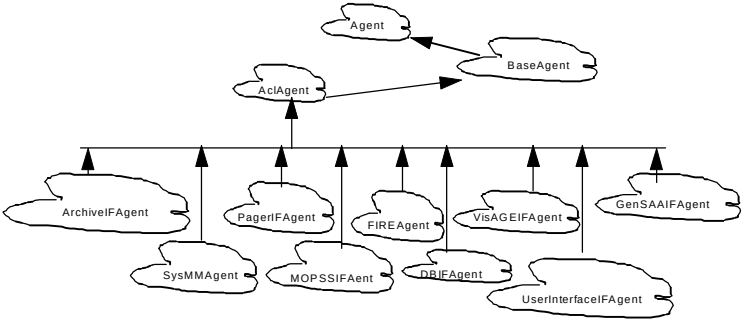

Figure 3.5. Class Hierarchy for LOGOS

Table 3.2. Class Definitions

Agent Class	Description
BaseAgent	BaseAgent defines a default implementation of the Agent interface that all LOGOS agents must support (such as clone, migrate, and runnable)
AclAgent	AclAgent encapsulates the agent-to-agent messaging components used by all agents. For the transport of messages the AclAgent uses a distributed software component framework called Workplace. WorkPlace provides the dynamic peer naming and data distribution services. The actual location of an agent and the transport protocols that are being used for communication are transparent to the agents.
SysMMAgent	Monitors all agents in the system in terms of system resource utilization, event timing and task completion.
MOPSSIFAgent	Monitors satellite pass schedule actibity and provides information concerning this schedule.
FIREAgent	Provides procedures for resolving reported mission faults or requests action by a human expert
VisAGEIFAgent	Serves as an interface to Visual Analysis Graphical Environment (VisAGE) which provides data/information visualization functions for spacecraft anomaly analysis and resolution, as well as general monitoring of the LOGOS Agent community as a whole
GenSAAIFAgent	Provides interfaces to GenSAA which is responsible for monitoring and controlling spacecraft behaviors, and to the FIREAent when an anomaly, not capable of being handled by GenSAA/Genie, is detected.
UserinterfaceIFAgent	Provides security through a user login process, translates messages from human readable form to the internal agent communication language, coordinates the writing of information on external user's displays by other agents, displays administrative or system-level messages, and maintains user-related information (e.g., prefernece, user modeling data, etc.)
DBIFAgent	Provides database management services for the LOGOS databases and is also responsible for the generation of any needed pass or anomaly reports.
ArchiveIFAgent	Responsible for maintaining an interface to an archive of recent and/or historical spacecraft telemetry data. When requested to do so, the Archive Interface Agent will retrieve specified telemetry points, perform any necessary processing on them, and forward them to the VisAGE Interface Agent for presentation to the LOGOS user.
PagerIFAgent	Pages human experts required for needed assistance

There are two agent types that are continuously present in LOGOS. These are the MOPSSAgent and the UserIFAgent. The MOPSSAgent awakens the SyaMMAgent in the event of an impending spacecraft pass, and the UserIFAgent does so if a remote user wishes to communicate with LOGOS. The capabilities supported by these agents are needed by ensure that there is always an interface to the environment outside of LOGOS and that the LOGOS can be readied in a timely fashion to support a mission contact and subsequent operations activities.

As an illustration of LOGOS activity consider the following high-level LOGOS scenario which highlights some of the major capabilities and activities within the system:

- Initialization of LOGOS to support spacecraft contact. The MOPSSIFAgent informs the SysMMAgent that a contact is coming up. The SysMMAgent ensures that the required hardware system elements are up and in good operating condition.
- The GenSAAIFAgent is activated. The SysMMAgent activates the GenSAAIFAgent. This agent brings up the GenSAA/Genie system, which provides for automated pass operations and initial detection of any spacecraft subsystem faults. The Genie script for the contact is activated. This script orchestrates the activities during the pass, including receiving and monitoring telemetry as well as commanding of the spacecraft.
- Once the contact begins, the TPOCC Data Server receives spacecraft telemetry and formats it for use by the GenSAA/Genie system.

- The GenSAAIFAgent monitors the behavior of the GenSAA/Genie system. If a fault is detected that cannot be handled by GenSAA/Genie system then the agent informs the FIREAgent for more extensive fault diagnosis.
- The FIREAgent attempts to diagnose the fault and recommend a solution. If it cannot, it makes the determination that outside help is required. It informs the UserIFAgent to contact the necessary analysts. The PagerIFAgent establishes the contact
- The VisAGEIFAgent is invoked to plan the method of data visualization that will be used to bring the contacted personnel up-to-date on the detected anomaly and to help establish the proper context for their decision making.
- The analysts diagnose the problem and initiate remedial actions through the UserIFAgent.
- The FIREAgent learns from the experience and updates its knowledge base to include information on the remedial actions identified by the analysts for the identified anomaly.
- If the analysts ask for a report on the situation the DBIFAgent will comply and support a report generation function.
- When time comes to end the contact, the SysMMAgent oversees closeout activities which includes data archiving, pass report generation, and shutting down of systems.

The scenario presented above illustrates several important ideas. It illustrates the role of agent as a tool user, it highlights interfaces and interactions with analysts outside the LOGOS on an as-needed basis (one of the hallmarks of lights-out operations), and it illustrates communication and cooperation among agents in the LOGOS community.

Agents within the LOGOS community communicate among themselves via an Agent Communication Language (ACL). This language is highly influenced by the KQML (Knowledge Query and Manipulation Language). In ACL there are currently four basic message types. These are: Request, Reply, Inform and Command. Each ACL message has a unique message id that is used for tracking and record keeping. Each message also includes sender-id, receiver-id, and return-to indicator among other fields. The list of performatives in the ACL includes: Give-Current -Status, Resolve-Anomaly, Query-DB, Insert-DB, Update-DB, Generate-Anomaly-Report, Generate-Pass-Report, Generate-Weekly-Report, Get-Mnemonic-Value, etc. New performatives are added as needs arise. In the new version of LOGOS the ACL will be KQML.

3.4 LOGOS - Next Steps

In addition to updating the agent communication language, new agent architecture will be introduced into the next instantiation of the agent testbed. (the current LOGOS will be used to provide the infrastructure for the new testbed.) The new agent architecture is a component-based architecture. The component approach will allow greater flexibility to the agent designer. A simple agent can be designed by using a minimum number of components that could receive percepts from the environment and react according to those percepts. This type of simple agent would be a reactive agent. A robust agent may be designed using more complex components that allow the agent to reason in a deliberative and/or social fashion, plan, schedule, model the environment, and learn. The following architecture focuses on an agent that is on the highly robust side of the scale; it contains components for modeling, reasoning, and planning. These components give the agent a higher degree of intelligence when interacting with its environment. Figure 3.5 details the agent architecture, showing the components needed to allow an agent to act with a high degree of intelligence.

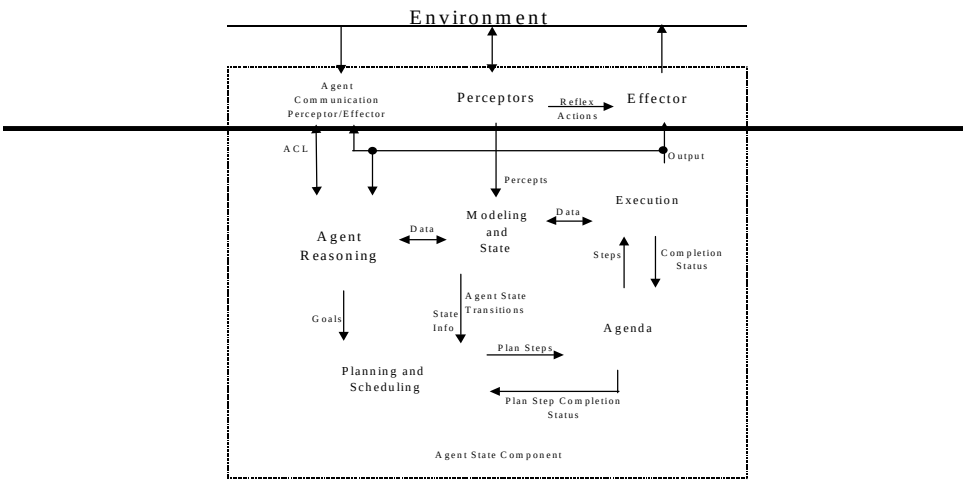

Figure 3.5. New Agent Architecture

Percepts received through sensors, communication with external software/systems, and other environmental entities are received through a Perceptor component. These percepts are passed from the perceptor to the Modeling and State component where a model's state is updated as needed. (Note, the modeling component maintains models of the environment, other agents in the community, and the agent itself.) A special perceptor is used to send and receive messages with other agents, this is the Agent Communication Perceptor/Effector. Incoming Agent Communication Language

(ACL) messages are formatted and passed to the Agent Reasoning component.. The Agent Reasoning component reasons with received ACL messages, knowledge that it contains, and information that is acquired from the Modeling and State component to formulate goals for the agent when necessary. Goals are then acquired by the Planning component along with state and state transition information. The Planning component formulates a plan for the agent to achieve the desired goals. When a plan has been developed, the Agenda keeps track of the execution of the plan's steps. Steps are marked when they are ready for execution and the completion status of each step is also tracked by the Agenda. The Execution component manages the execution of steps and determines the success or failure of each step's execution. Output produced during a step execution can be passed to an Effector or the Reasoning component. The Modeling and State component will record state changes. When a plan is finished execution, a completion status is passed to the Planning component for evaluation of the plan for future use. If the Agent Reasoning component is dealing with data from the environment it may decide to either set a goal (for more deliberative planning) or react quickly in an emergency situation. The Agent Reasoner can also carry on a dialog with another agent in the community through the Agent Communication Perceptor/Effector.

What is a component? A component is a software module that performs a defined task. Components when combined with other software components can constitute a more robust piece of software that is easily maintained and upgraded. Each component in the architecture can communicate information to/from all other components as needed through a publish and subscribe communication mechanism, message passing, or a request for immediate data. Components may be implemented with a degree of intelligence through the addition of reasoning and learning functions in each component. Thus, a component needs to implement certain interfaces and contain certain properties. Components must implement functionality to publish information, subscribe to information, and be able to accept queries for information from other components or objects. Components need to keep a status of their state, and need to know what types of information they contain and need from external components and objects to function.

The LOGOS experience has to-date had been a successful and rewarding one. Because of its initial success longer-range missions like the Nanosat mission, a constellation of small satellites which will make multiple remote and in-situ measurements in space, has plans to evaluate the role that agent technology can play in support of autonomous ground and space operations.

4. Remote Agent

We now turn from ground-based autonomy to autonomy onboard a spacecraft. This section describes a flight experiment that will demonstrate the Remote Agent approach to spacecraft commanding and control. In this approach, the Remote Agent software may be considered to be an autonomous "remote agent" of the spacecraft operators in the sense that the operators rely on the agent to achieve particular goals. The operators do not know the exact conditions on the spacecraft, so they do not tell the agent exactly what to do at each instant of time. They do, however, tell the agent exactly which goals to achieve in a period of time as well as how and when to report in.

The Remote Agent (RA) is formed by the integration of three separate technologies: an on-board planner-scheduler, a robust multi-threaded executive, and a model-based fault diagnosis and recovery system. This Remote Agent approach is being designed into the New Millennium Program's Deep Space One (DS1) mission as an experiment. The New Millennium Program is designed to validate high-payoff, cutting-edge technologies to enable those technologies to become more broadly available for use on other NASA programs. The experiment is slated to be exercised in May of 1999.

4.1 Remote Agent Design Approach and Architecture

The New Millennium Autonomy Architecture rapid Prototype (NewMaap) effort [3] identified the key contributing technologies: on-board planning and replanning, multi-threaded smart executive, and model-based failure diagnosis and repair. In NewMaap, we learned how to take advantages of the strengths and weaknesses of these three technologies and merge them into a powerful system. After successful completion of the prototype, the RA was selected as one of the NMP technologies for DS1. It will be uplinked to the spacecraft as a software modification and demonstrated as an experiment.

Figure 4.1 shows the communications architecture for the Remote Agent. Note that all interactions with the hardware are the responsibility of the real-time software. The RA is layered on top of that software, but also gathers information from all levels to support fault diagnosis.

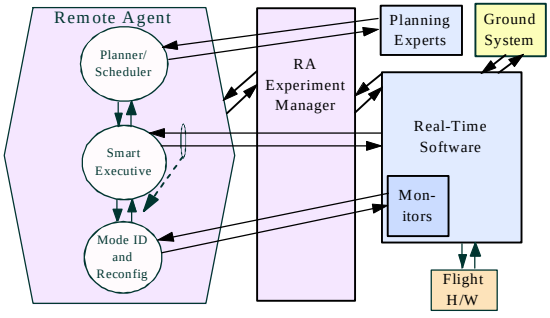

Figure 4.1. Remote Agent Communication Architecture

Several spacecraft commanding styles are possible. Goal-oriented commanding is the intended operating mode for most of a RA mission; provision has been made for updating the goals in flight. In a typical planning cycle, the executive is executing a plan and gets to an activity that can be interpreted as "time to plan the next segment." The executive calls the planner with the current and projected spacecraft state including the health of all devices. The planner/scheduler generates a new plan using priorities, heuristics, and domain models including system constraints. The planner sends this plan to an executive that creates an agenda of plan items and executes the agenda. Plan execution robustness is added by making use of the Model-based Mode Identification and Reconfiguration (MIR) system. The MIR system includes monitors, mode identification for nominal and failure conditions, communication of state to the executive and proposals of reconfiguration actions to take in the event of failures. Each of the components of the Remote Agent will be described in more detail, but first the Remote Agent experiment for the Deep Space One mission will be described in more detail.

4.2 The Deep Space One Remote Agent Experiment

The Remote Agent eXperiment (RAX) for Deep Space One is a demonstration of RA capabilities. Since an alternate method of control is used for most of the mission, RAX is focused on demonstrating specific autonomy capabilities rather than controlling all aspects of spacecraft behavior. The Remote Agent controls the following spacecraft hardware and software: the camera for use in autonomous navigation, the Solar Electric Propulsion (SEP) subsystem for trajectory adjustment, the attitude control system for turns and attitude hold, the navigation system for determining how the actual trajectory is deviating from the reference trajectory and what SEP thrusting profile is needed to stay on the reference trajectory, and the Power Amplification and Switching Module (PASM) for use in demonstrating fault protection capabilities.

Four failure modes are covered by RAX. These are:

F1. Power bus status switch failure

F2. Camera power stuck on

F3. Hardware device not communicating over bus to flight computer

F4. Thruster stuck closed

4.3 Mission Scenario

The Remote Agent experiment is executed in two phases, a 12 hour Phase One followed a couple of weeks later by a several day Phase Two.

In Phase One, we start slowly by first demonstrating the executive operating in the manner of a low-level sequencer by accepting commands to turn devices on and off. Next, a "scripted" mode is demonstrated with execution of plans uplinked from the ground. The main demonstration here will be commanding the spacecraft to go to and stay in a known, safe, standby mode and then take a series of optical navigation (OpNav) images.. In addition, Failure mode F1 will be demonstrated by injecting power bus switch status readings indicating that a power bus is unexpectedly off. The fault diagnostic system will examine this information along with other information that indicates that devices on the bus are still communicating normally with the flight computer and conclude that the failure is in the switch status sensor and not in the bus itself. No action will result

In Phase Two, we initiate on-board planning. Based on the spacecraft initial state and the uplinked goals, the planner will generate a multi-day plan including imaging for optical navigation, thrusting to stay on the reference trajectory, and simulated injection of faults to test out failures F2, F3, and F4. First the camera power stuck on failure (F2) is injected. When the executive is unable to turn off the camera when the plan so dictates, the executive realizes that the current plan should be aborted and replanning is indicated. This might be necessary, for example, because the initial plan's assumptions on power consumption are incorrect with the camera on when it should be off. The plan is declared failed, the spacecraft is sent to a standby mode while the planner is requested to replan based on the new information that the camera power switch is stuck on. When the new plan is received by the executive, execution resumes including navigation and SEP thrusting. Near the end of the three-day plan, the planner is called to generate the plan for the next three days. This plan includes navigation and SEP thrusting as before. It also includes two simulated faults. First, a failure of a hardware device to communicate is injected (F3); the proper recovery is to reset the device without interrupting the plan. Next, injecting an attitude control error monitor above threshold simulates a thruster stuck closed failure (F4). The correct response is to switch control modes so that the failure is mitigated.

4.4 RA Capabilities Demonstrated with DS1 RAX

The above scenario has been designed to demonstrate that the DS1 Remote Agent meets the following autonomy technology goals:

- Allow low-level command access to hardware.
- Achieve goal oriented commanding.
- Generate plans based on goals and current spacecraft state expectations.
- Determine the Health State of hardware modules.
- Demonstrate model-based failure detection, isolation, and recovery.
- Coordinate hardware states and software modes.
- Replan after failure given a new context.

4.5 RA Components

The major components of the Remote Agent are discussed as follows.

4.5.1 Planner/Scheduler

The highest level commanding interface to the Remote Agent is provided the Planner/Scheduler (PS). Given the Spacecraft State and a set of goals, PS generates a set of synchronized high-level activities that, once executed, will achieve the goals. PS maintains a database of goals for the mission, and the mission profile that spans a very long time horizon, potentially the duration of the entire mission. Ground controllers can add, modify, or delete goals from the mission profile by issuing a command to Remote Agent.

Over the duration of a mission PS is invoked by the executive to return a synchronized network of activities, the plan, for each short-term scheduling horizon into which the mission is partitioned. Typically each short-term horizon covers several days. When PS receives a request from EXEC, it identifies the next scheduling horizon and retrieves the goals relevant to that horizon from the mission profile. It merges in the expected initial Spacecraft State provided by EXEC and generates a fully populated plan. PS then sends that plan to EXEC for execution. A new plan will be requested in two situations, both of which are handled identically by PS:

- Nominal operations: in this case EXEC reaches the activity Planner_Plan_Next_Horizon toward the end of the current plan horizon. EXEC will issue a request for a new plan. This request will define the new initial state as the expected final state of the current plan. This will allow transition from the old to new plan without any interruption of execution.

- Fault response: if MIR detects an anomaly that will impact the executability of current or future tasks in the plan, the EXEC will request a new plan to resume normal operations after having put the spacecraft in a safe standby mode. In this case the initial state describes the standby state for each subsystem modeled in the plan and health information describing possibly degraded modes for failed subsystems.

PS consists of a heuristic search engine, the Incremental Refinement Scheduler (IRS) that operates in the space of incomplete or partial plan [6]. Since the plans explicitly represent time in a numeric (or metric) fashion, the planner makes use of a temporal database. As with most causal planners, PS begins with an incomplete plan and attempts to expand it into a complete plan by posting additional constraints in the database. These constraints originate from the goals and from constraint templates stored in a model of the spacecraft. The temporal database and the facilities for defining and accessing model information during search are provided by the HSTS system. For more details on PS and the HSTS system see [4] and [5]. Figure 4.2 illustrates the PS architecture.

Each subsystem in the model is represented in the PS database. Each subsystem has a set of dynamic state variables whose value is tracked over time. PS uses tokens to represent both actions and states that occur over finite time intervals. A token is associated with a value of a state variable occurring over a finite time interval. Each value has one or more associated compatibility's, i.e., and sets of constraints between tokens. An example of the atomic temporal constraints that belong to compatibility can be expressed in English as “While the spacecraft is taking asteroid pictures requested by navigation, no ion thrusting is allowed”.

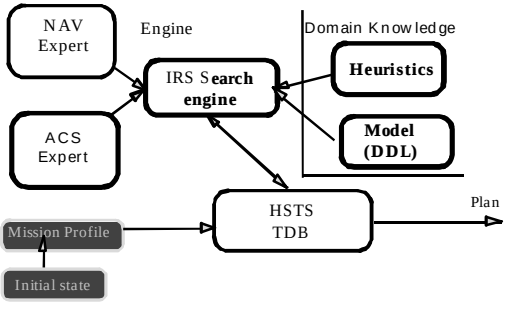

Figure 4.2. Planner/Scheduler Architecture

Table 4.2 describes three plans that PS would generate during a 6-day period that includes replanning due to spacecraft failure. The first CPU time column reports the actual run time of PS on a PowerPC/VxWorks flight hardware testbed. The next column reports the estimated time to generate the same plans using only 25% of DS1’s slower RAD6000 processor.

Scenario	Tokens	Constraints	CPU time on PPC testbed (mm:ss)	Est. CPU time on RAD6000 (hh:mm:ss)
1st horizon	105	141	7:13	4:48:00
Replan in 1st horizon	69	66	4:01	2:40:00
2nd horizon	126	192	13:49	9:12:00

Table 4.2 PS Metrics for Performance

4.5.2 Executive

The Smart Executive (EXEC) is a reactive plan execution system with responsibilities for coordinating execution-time activity. EXEC's functions include plan execution, task expansion, hardware reconfiguration, runtime resource management, plan monitoring, and event management. The executive invokes the planner and MIR to help it perform these functions. The executive also controls the lower-level software by setting its modes, supplying parameters and by responding to monitored events.

The top-level operational cycle, including the planning loop, is described as follows. EXEC requests a plan by formulating a plan request describing the current execution context.. It later executes and monitors the generated plan. EXEC executes a plan by decomposing high-level activities in the plan into primitive activities, which it then executes by sending out commands, usually to the real-time flight software (FSW). EXEC determines whether its commanded activities succeeded based either on direct feedback from the recipient of the command or on inferences drawn by the Mode Identification (MI) component of MIR. When some method to achieve a task fails, EXEC attempts to accomplish the task using an alternate method in that task's definition or by invoking the Mode Reconfiguration (MR) component of MIR. If MR finds steps to repair the failing activity without interfering with other concurrent activities, EXEC performs those steps and continues on with the original definition of the activity. If the EXEC is unable to execute or repair the current plan, it aborts the plan, cleans up all executing activities, and puts the controlled system into a standby mode. If continued autonomous operation is allowed, EXEC requests a new plan from PS while maintaining standby mode. EXEC resumes execution of the new plan when it is received.

In the Periodic Planning Cycle as shown in Figure 4.3, our approach separates an extensive, deliberative planning phase from the reactive execution phase, executing infrequently generated plans that extended over time periods. Plans normally include the task of planning for the next horizon—i.e., the planner sets aside a good time for its own (next) computation. The planner extends the plan to address the goals of the next planning horizon and returns the result to the executive. The executive then merges the extended plan with the end of the currently running plan. This enables Remote Agent to engage in activities, which span multiple planning horizons (such as a 3-month long ion engine burn), without interrupting them.

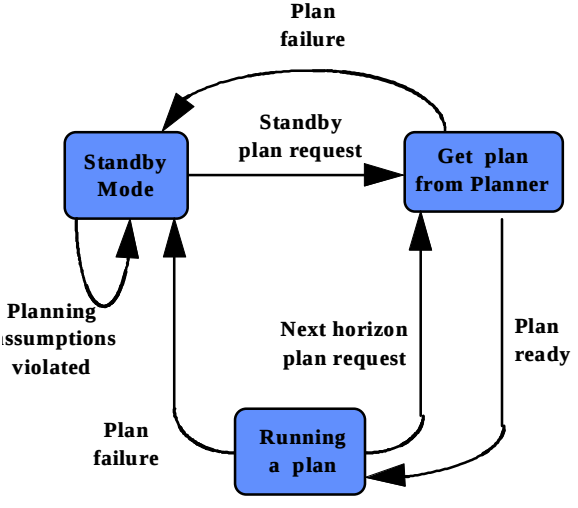

Figure 4.3 Executive Periodic Planning Cycle

We now summarize how the EXEC capabilities described above are demonstrated within the RAX scenarios.

First, EXEC demonstrates the multi-level commanding, allowing ground operators to specify low-level commands to the hardware as part of a sequence, to generate plans from ground, or to request and execute plans generated on-board the spacecraft. Low-level commanding and ground-based planning are demonstrated in Phase One of the RAX experiment. In this phase a plan is up-linked from the ground which contains both high-level activities (like turning to a target) and low-level activities (using the EXEC-ACTIVITY tokens turn PASM on and off).

Second, EXEC demonstrates plan request generation and execution. As part of executing a plan phase two, EXEC demonstrates a number of important capabilities involved in token decomposition.

- EXEC demonstrates context sensitive behavior in the management of the ion propulsion system. Before executing a thrust command, EXEC requires that IPS be in standby mode. If it is already in standby mode, EXEC proceeds to the thrusting, otherwise it will put IPS into the standby mode before proceeding.

- EXEC demonstrates time-driven token durations. For example, it terminates a thrust segment based on a timeout, rather than external confirmation.
- EXEC demonstrates event-driven token duration's, in which picture taking tokens are not allowed to terminate until confirmation the picture has finished
- EXEC demonstrates goal-oriented achievement (don't achieve things that are already true). Because the planner is unable to determine how many thrust segments are necessary to achieve the total desired thrust, it inserts thrust tokens into the plan, which may not need to be executed. EXEC tracks how much thrust has been achieved, and only executes thrust tokens (and associated turns) for so long as thrust is actually necessary..
- EXEC demonstrates the coordination of activity details across subsystems that are below the level of visibility of the planner. There is a constraint that ACS is in that thrust-vector-control (TVC) mode shortly after IPS has started thrusting. When EXEC commands IPS into thrusting mode, it also sends the command to ACS to enter TVC mode based on its own lower-level domain knowledge.

Third, EXEC demonstrates the ability to maintain required properties in the face of failures. In the thruster failure scenario, EXEC learns from a MIR state update that the current thruster mode is faulty. It invokes MIR with a recovery request and then executes MIRs recommendation to change to a degraded thruster control mode.

Fourth, EXEC demonstrates the ability to recognize plan failure, abort the plan, enter standby mode, and request and execute a replan. This occurs in the MICAS (Microelectronics Integrated Camera And Spectrometer) failure scenario, in which EXEC learns from MIR that MICAS is stuck on and cannot be turned off. EXEC requests a recovery from MIR so that it can turn MICAS off, but since there is no way to fix this problem MIR informs EXEC that it has no recovery. Since the plan requires MICAS to be off, EXEC aborts the plan, terminating a thrusting segment if necessary. It then enters a degraded standby mode, in which it leaves MICAS on despite the usual desire to turn off all unnecessary devices in standby mode, and requests a plan for the planner. In its plan request, EXEC informs the planner that MICAS is stuck on. Later, in executing the new plan, ground finds a way to fix MICAS and informs MIR of this fact. When EXEC learns from MIR that MICAS can now be shut off, this new information does not cause EXEC to abandon the plan, since the planner did not require MICAS to be broken. However, the next time EXEC asks for a plan, it informs the planner about the restored health of MICAS, so that the planner can now plan to switch MICAS off when desired. More details about EXEC can be found in References [6, 7... and 8].

4.5.3 Diagnosis and Repair

We refer to the Diagnosis and Repair engine of the Remote Agent as MIR, for Mode Identification and Reconfiguration, which emphasizes the model-based diagnosis and control flavor of the system. MIR eavesdrops on commands that are sent to the on-board hardware managers by the EXEC.. As each command is executed, MIR receives observations from spacecraft sensors. MIR combines these commands and observations with declarative models of the spacecraft's components to determine the current state of the system and report it to the Exec. A very simple example is shown schematically in Figure 4.4. In the nominal case, MIR merely confirms that the commands had the expected effect on Spacecraft State. In case of failure, MIR diagnoses the failure and the current state of the spacecraft and provides a recovery recommendation. A single set of models and algorithms are exploited for command confirmation, diagnosis and recovery.

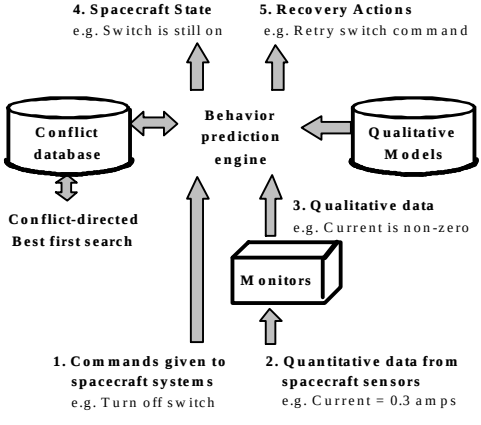

Figure 4.4 Information Flow in MIR

The RAX mission scenario demonstrates the following MIR capabilities: state identification throughout the experiment, diagnosis of sensor failure, diagnosis and recovery recommendations for device failures, and overriding of a MIR diagnosis via a ground command. RAX is fully responsible for responding to real failures within scope should they occur during the experiment. However, since we cannot depend on failures occurring during the experiment, injecting false sensor readings consistent with the failures will simulate failures F1-F4. RAX will be expected to take the appropriate corrective actions, though in reality none are necessary.

F1 illustrates MIR's ability to disambiguate between a sensor failure and failure of the device being sensed. MIR combines power distribution models with the sensed nominal current draw and communication status of devices to conclude that the power switch must be on and that a switch sensor failure, though unlikely, has occurred. Failures F2-F4 are diagnosed in a similar fashion and include the

possibility of recovery. In F4, given only an attitude error and models of the spacecraft dynamics, MIR infers that one of a particular pair of thruster valves is stuck closed. MIR is then able to recommend that no matter which one of the two valves is stuck, switching ACS control modes will mitigate the problem.

The MIR component of the RA architecture, embodied in a system called Livingstone, consists of two parts: Mode Identification (MI) and Mode Reconfiguration (MR). MI is responsible for identifying the current operating or failure mode of each component in the spacecraft. Following a component failure, MR is responsible for suggesting reconfiguration actions that restore the spacecraft to a configuration that achieves all current goals as required by the planner and executive. Livingstone can be viewed as a discrete model-based controller in whom MI provides the sensing component and MR provides the actuation component. MI's mode inference allows the executive to reason about the state of the spacecraft in terms of component modes, rather than in terms of low level sensor values, while MR supports the run-time generation of novel reconfiguration actions.

Livingstone uses algorithms adapted from model-based diagnosis [9, 10] to provide the above functions. The key idea underlying model-based diagnosis is that a combination of component modes is a possible description of the current state of the spacecraft only if the models associated with these modes are consistent with the observed sensor values. Following de Kleer and Williams [10], MI uses a conflict directed best-first search to find the most likely combination of component modes consistent with the observations. Analogously, MR uses the same search to find the least-cost combination of commands that achieve the desired goals in the next state. Furthermore, both MI and MR use the same system model to perform their function. The combination of a single search algorithm with a single model, and the process of exercising these through multiple uses, contributes significantly to the robustness of the complete system. Note that this methodology is independent of the actual set of available sensors and commands. Furthermore, it does not require that all aspects of the spacecraft state are directly observable, providing an elegant solution to the problem of limited observability.

The use of model-based diagnosis algorithms immediately provides Livingstone with a number of additional features. First, the search algorithms are sound and complete, providing a guarantee of coverage with respect to the models used. Second, the model building methodology is modular, which simplifies model construction and maintenance, and supports reuse.. Third, the algorithms extend smoothly to handling multiple faults and recoveries that involve multiple commands. Fourth, while the algorithms do not require explicit fault models for each component, they can easily exploit available fault models to find likely failures and possible recoveries.

An important Livingstone feature is that the behavior of each component state or mode is captured using abstract, or qualitative, models [11, 12]. These models describe qualities of the spacecraft's structure or behavior without the detail needed for precise numerical prediction, making abstract models much easier to acquire and verify than quantitative engineering models. Examples of qualities captured are the power, data and hydraulic connectivity of spacecraft components and the directions in which each thruster provides torque. While such models cannot quantify how the spacecraft would perform with a failed thruster for example, they can be used to infer which thrusters are failed given only the signs of the errors in spacecraft orientation. Such inferences are robust since small changes in the underlying parameters do not affect the abstract behavior of the spacecraft. In addition, abstract models can be reduced to a set of clauses in propositional logic. This form allows behavior prediction to take place via unit propagation, a restricted and very efficient inference procedure.

All told, the MIR model for the DS1 spacecraft represents fifty-seven components of twelve different types, their behavior, and their interconnections. It is important to note that the MIR models are not required to be explicit or complete with respect to the actual physical components. Often models do not explicitly represent the cause for a given behavior in terms of a component's physical structure. For example, there are numerous causes for a stuck switch: the driver has failed, excessive current has welded it shut, and so on. If the observable behavior and recovery for all causes of a stuck switch are the same, MIR need not closely model the physical structure responsible for these fine distinctions. By modeling only to the level of detail required to make relevant distinctions in diagnosis (distinctions that prescribe different recoveries or different operation of the system) we can describe a system with qualitative "common-sense" models which are compact and quite easily written.

Models are always incomplete in that they have an explicit unknown failure mode. Any component behavior, which is inconsistent with all known nominal and failure modes, is consistent with the unknown failure mode. In this way, MIR can infer that a component has failed, though the failure was not foreseen or was simply left unmodeled because no recovery is possible. Additional technical details about Livingstone can be found in [13]

5. Future Directions - Some Speculations

Extrapolating from the LOGOS and Remote Agent examples, its safe to say that the future of agent technology work in NASA is exceptionally bright. A major accomplishment will be the integrating of ground-based and space-based autonomous systems into a single autonomous system with a ground and a space component. This integration could support some interesting and valuable investigations and demonstrations. For example, the migration of an autonomous process from the ground to space could be viewed as an agent moving from one

community to another. This would be an interesting way to achieve adaptable autonomy.

In closing we cite three scenarios that are in NASA's future:

- A Principal Investigator (PI) talking, in a natural language, to an onboard agent serving as a PI-surrogate which is responsible for the management of the PIs scientific instrument and the planning and scheduling of detailed approaches for realizing the science agenda established by the PI.
- The spacecraft itself, considered as an agent, initiating and maintaining a dialog with an automated ground system and/or a human for either status reporting or anomaly handling.
- A constellation of autonomous spacecraft deciding among themselves how best to achieve some scientific goal.

6. Acknowledgments

At Goddard the authors wish to thank James Rash, Tom Grubb, Troy Ames, Carl Hostetter, Chris Rouff, Jeff Hosler, Matt Brandt, Dave Kocur, Kevin Stewart, Jay Karlin, Victoria Yoon, Chariya Peterson, and Dave Zock (who are or have been members of the Goddard Agent Group) for their major contributions. For more information on LOGOS the interested reader is invited to browse the GSFC agent web page at http://agents.gsfc.nasa.gov for access to current papers and working documents.

At ARC and JPL the authors wish to thank Douglas E. Bernard[1], Gregory A. Dorais[3], Chuck Fry[3], Edward B. Gamble Jr. [1], Bob Kanefsky[3], James Kurien[5], William Millar[3], Nicola Muscettola[2], P. Pandurang Nayak[4], Barney Pell[4], Kanna Rajan[3], Nicolas Rouquette[1], Benjamin Smith[1], Brian C. Williams[5] ([1] Jet Propulsion Laboratory, [2] Recom Technologies @ Ames, [3] Caelum Research @ Ames, [4] RIACS @ Ames

The material on the Remote Agent was based on a paper presented at IEEE Aerospace98 Conference

7. References

[1] Goddard Agent Group Working Paper, "Intelligent Agent-based Systems in the NASA/GSFC Context", http://agents.gsfc.nasa.gov/products.html

[2] Goddard Agent Group Working Paper, "LOGOS Requirements and Design Document", http://agents.gsfc.nasa.gov/products.html

[3] B. Pell, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. Nayak, M. D. Wagner, and B. C. Williams, “A Remote Agent Prototype for Spacecraft Autonomy,” SPIE Proceedings Volume 2810, Denver, CO, 1996.

[4] N. Muscettola, “HSTS: Integrating planning and scheduling,” in Fox, M., and Zweben, M., eds, *Intelligent Scheduling*, Morgan Kaufman,

[5] N. Muscettola, B. Smith, S. Chien, C. Fry, G. Rabideau, K. Rajan, D. Yan, “Onboard Planning for Autonomous Spacecraft,” in Proceedings of the fourth International Symposium on Artificial Intelligence, Robotics and Automation for Space (i-SAIRAS 97), July 1997.

[6] Barney Pell, Ed Gamble, Erann Gat, Ron Keesing, Jim Kurien, Bill Millar, P. Pandurang Nayak, Christian Plaunt, and Brian Williams, “A hybrid procedural/deductive executive for autonomous spacecraft,” In P. Pandurang Nayak and B. C. Williams, editors, Procs. of the AAAI Fall Symposium on Model-Directed Autonomous Systems, AAAI Press, 1997.

[7] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith, “Robust Periodic Planning and Execution for Autonomous Spacecraft,” In Procs. of IJCAI-97, 1997.

[8] Erann Gat and Barney Pell, “Abstract Resource Management in an Unconstrained Plan Execution System,” in Proc. of IEEE Aeronautics (AERO-98), Aspen~~d~~, CO, IEEE Press, 1998 (To appear).

[9] J. de Kleer and B. C. Williams, “Diagnosing Multiple Faults," Artificial Intelligence, Vol 32, Number 1, 1987.

[10] J. de Kleer and B. C. Williams, “Diagnosis With Behavioral Modes," Proceedings of IJCAI-89, 1989.

[11] D. S. Weld and J. de Kleer, *Readings in Qualitative Reasoning About Physical Systems*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[12] J. de Kleer and B. C. Williams, *Artificial Intelligence*, Volume 51, Elsevier, 1991.

[13] B. C. Williams and P. Nayak, “A Model-based Approach to Reactive Self-Configuring Systems,” Proceedings of AAAI-96, 1996.